ARMY RESEARCH LABORATORY

Migrating Dari Clustergen Flite Text-to-Speech Voice from Desktop to Android

by Michael H Lee

ARL-TN-0632 September 2014

Army Research Laboratory
Adelphi, MD 20783-1138

ARL-TN-0632 **September 2014**

Migrating Dari Clustergen Flite Text-to-Speech Voice from Desktop to Android

Michael H Lee
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2014	**2. REPORT TYPE**	**3. DATES COVERED (From - To)** 03/2014–05/2014
4. TITLE AND SUBTITLE Migrating Dari Clustergen Flite Text-to-Speech Voice from Desktop to Android		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER R.0010385.15
6. AUTHOR(S) Michael H Lee		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-CII-B 2800 Powder Mill Road Adelphi, MD 20783-1138		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-0632
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Festival Lite (Flite) is a compact version of Festival, a speech synthesis engine developed by Dr Alan W Black of Carnegie Mellon University (CMU). When Flite is initially downloaded, it is configured to compile and operate as a standalone program on a variety of platforms (e.g., Linux, Windows, Cygwin, etc.). In 2010, Dr Alok Parlikar, a CMU researcher/developer, developed an Android platform support layer for Flite. This report describes the process of adding a Dari Flite Clustergen voice to the Flite Android engine.

15. SUBJECT TERMS
Festival Lite, Flite, Clustergen, Text-to-Speech, TTS, Android

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Michael H Lee
A. Report	**b. ABSTRACT**	**c. THIS PAGE**	UU	44	**19b. TELEPHONE NUMBER** (*Include area code*)
Unclassified	Unclassified	Unclassified			(301) 394-5608

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

In 2012, the US Army Research Laboratory (ARL) developed an Android app called the Medical Phrasebook to serve as a bilingual English-Dari glossary of more than 6000 technical medical terms.[1] Since then, new enhancements (e.g., Pashto texts, military terms, additional Dari terms, etc.) were added to the Phrasebook app. In 2014, we wanted to add text-to-speech (TTS) capability to the app. Implementing an English TTS was trivial because a mature English TTS was available in the Android's TTS application programming interface (API). Not surprisingly, neither Dari nor Pashto was included in the Android's TTS API. After searching in academia and the research community, we found an open-source TTS Engine called Festival Lite with Dari language support developed by Carnegie Mellon University (CMU).

Festival Lite (Flite) is a smaller footprint version of a speech synthesis framework called Festival. Flite is commonly used by developers to create custom TTS engines. It works well as a standalone application in Linux and Windows. It is easy to use, requires few resources, and synthesizes voices very quickly. To provide Flite to a wider audience, Flite has been ported to the Android platform by an open source project called the Flite Android Engine.[2] The Flite Android Engine does not attempt to rewrite the Flite engine in Android; it is simply a Google TTS wrapper for Flite and actually requires Flite to be present at compile time. The Flite Android Engine has a user interface that allows users to download and select voices from a list and also a demo page that lets a user synthesize a voice from text.

We searched for instructions on how to integrate the Dari TTS library to the Flite Android Engine, but there were few resources describing this process and some posted instructions did not produce expected results. This integration was not an intuitive task due to the fact that the Flite Android Engine did not have an easy way to manually integrate new language (e.g., Dari, Pashto) libraries. With a continuously evolving state and various development versions of Flite, we encountered various obstacles migrating the Flite Dari library to the Android platform. Our issues may have been related to the specific Flite Engine version we worked with. The latest available version of Flite Engine is 1.9.0, while the Flite Android Engine instructions describe integrating with Flite Engine, version 1.5.6.

Depending on the version of Flite and the Flite language to be integrated, other developers may not encounter the same issues we describe in this report. At the time of our migration work, there was no comprehensive documentation explaining this process. Many troubleshooting emails were sent back-and-forth with Dr Alan Black (the primary developer of Flite) and Dr Alok Parlikar (the primary developer of the Flite Android Engine) to resolve these issues. In this report, we document the information learned from the email exchanges with the Flite Engine and Flite Android Engine developers, a step-by-step process of migrating the Flite Dari library from a desktop platform to an Android platform, and finally, the process of how we integrated the Flite Dari TTS capability to the ARL Phrasebook Android app.

2. Two Types of Dari Voices – Clunit and Clustergen

There are two versions of Dari voices available from CMU: Clunit and Clustergen. Clunit (Unit Select)[3] and Clustergen (statistical parametric synthesis)[4] are two TTS voice building/synthesis techniques. Both voices work very well as standalone applications on Linux and Windows. According to the ARL's in-house Dari linguist, Ghulam Jahed, the Dari Unit Select voice was superior to Dari Clustergen voice—Dari Unit Select sounded more natural than Dari Clustergen.

Unfortunately, this report does not focus on Dari Unit Select, because we were not able to successfully migrate it to Android due to a very specific "Libc Fatal Signal" error thrown by the Dari Unit Select library. According to Dr Parlikar, the migration process for both types of voices is very similar, and most of the instructions described here are applicable to Dari Unit Select as well.

Even though we do not address Dari Unit Select in this report, we provide useful information (when appropriate to the topic of discussion) that pertains specifically to Dari Unit Select that we gathered from our effort that may help other developers who attempt to migrate Dari Unit Select to an Android platform.

3. Enabling Dari on Flite Engine 1.5.6 and Compiling Flite Engine 1.5.6

Our interest was in specifically migrating Dari to the Android platform. Dr Black provided us with a 29-MB Flite, version 1.9.0 (configured for Dari and Pashto), and two variations of the Dari voice, a 4.12-GB Unit Select and a 784-MB Clustergen. (Note: We were told the migration process to Android was the same for Unit Select and Clustergen; however, we were not able to migrate the Unit Select voice to Android.)

We were able to set up Flite 1.9.0 and synthesize both Dari voices as a standalone application on Windows and Linux. Later during the migration effort, we ran into a software bug specific to Flite, version 1.9.0. Although the Android Flite Engine had basic functionality (e.g., display About Flite, download voices, set voices, switching system TTS to Flite, and Demo GUI, etc.), it was not able to "set language" using the Flite 1.9.0 native library. Without the ability to set the language, the Android Flite Engine could not synthesize any voice. We circumvented this software bug by working with Dr Black to extract the Dari capability from Flite 1.9.0 and integrate it to Flite 1.5.6, which was known to be compatible with the Android Flite Engine. This extraction and integration process only *enables* the Flite 1.5.6 engine to link with Dari voices.

The Dari voices then have to be compiled separately. We describe the Dari extraction and integration process in this section.

1. **Download Flite 1.9.0 and 1.5.6.**

   Please contact Dr Alan Black at CMU[5] for instructions on downloading Flite 1.9.0 and the Dari voice libraries. Flite, version 1.5.6, is available for download from CMU TTS Web page.[6] This report focuses on migrating Dari to Android, but the version of Flite 1.9.0 we received was configured for Dari and Pashto. As we extracted the Dari capability from Flite 1.9.0, we extracted Pashto as well.

2. **Extract Flite 1.9.0 and 1.5.6 to disk.**

   For basic setup and standalone operation, the Flite TTS Engine can be compiled on either Linux or Cygwin (i.e., Windows). But we recommend performing these steps on a Linux system from the beginning because certain steps that are required later in the Android migration effort do not work in Cygwin and require in-depth troubleshooting. It is much easier to develop and proceed this application in Linux. We chose to develop on a 64-bit Ubuntu Linux. Perform the following steps:

   - Extract Flite 1.9.0 to "/Flite/flite-1.9.0-current".
   - Extract Flite 1.5.6 to "/Flite/flite-1.5.6-current_w_1.9.0_Dari_Pashto".

   For the remainder of this report, "/Flite/flite-1.5.6-current_w_1.9.0_Dari_Pashto" is referenced as <FLITE_HOME>.

3. **Copy 1.9.0's Dari and Pashto "lang" files to 1.5.6.**

   Perform the following:

```
cp     /Flite/flite-1.9.0-current/lang/cmu_dari_lang/  <FLITE_HOME>/lang
cp     /Flite/flite-1.9.0-current/lang/cmu_dari_lex/   <FLITE_HOME>/lang
cp     /Flite/flite-1.9.0-current/lang/cmu_pashto_lang/<FLITE_HOME>/lang
cp     /Flite/flite-1.9.0-current/lang/cmu_pashto_lex/ <FLITE_HOME>/lang
cp     /Flite/flite-1.9.0-current/lang/cmu_us_kal/     <FLITE_HOME>/lang
cp     /Flite/flite-1.9.0-current/lang/cmu_us_rms/     <FLITE_HOME>/lang
```

Although "cmu_us_kal" and "cmu_us_rms" are English voices, they must be copied to Flite 1.5.6 as well. Otherwise, the directories' absence will cause an error during the make process.

4. **Configure and compile Flite 1.5.6 Engine.**

   Perform the following:

   Copy "transtac.lv" from Flite 1.9.0 if it does not already exist in Flite 1.5.6.

```
cp     /Flite/flite-1.9.0-current/config/transtac.lv
 <FLITE_HOME>/config
```

Generate a new “config” file for Flite 1.5.6.

The Dari configured Flite TTS Engine we used required a special parameter. Use the “--with-langvox=transtac” parameter to generate a new “config” file and compile Flite 1.5.6.

```
cd <FLITE_HOME>
./configure --with-langvox=transtac
make
```

This command will generate a “<FLITE_HOME>/config/config” file with the correct “langvox” value we need.

After successful compilation, there will be new binaries in the “<FLITE_HOME>/bin” folder.

Flite 1.5.6 is now able to link with a Dari voice. Linking a Dari Clustergen voice to this Flite 1.5.6 installation is described in the next section. However, Flite 1.5.6 is shipped with the ability to synthesize English text immediately after compilation.

5. **Verify Flite TTS Engine with English voice synthesis.**

If Flite TTS Engine was installed correctly, it should speak the following text:

```
cd <FLITE_HOME>
./bin/flite "Flite is a small fast run-time synthesis engine"
input_text Flite is a small fast run-time synthesis engine
```

On Linux, the above command may display an error message if the Open Sound System (OSS) package is not available:

`“oss_audio: failed to open audio device /dev/dsp”`.

This error message does not indicate a problem with Flite TTS Engine and is remedied by installing OSS on the target Linux platform. Unless OSS is set up in Linux, a synthesized voice will not play.

An alternative way to test Flite is to generate a WAV file (8 KHz riff headered waveform) of the text:

```
cd <FLITE_HOME>
./bin/flite "Flite is a small fast run-time synthesis engine"
testing.wav
input_text Flite is a small fast run-time synthesis engine
```

This command generates “testing.wav” in the <FLITE_HOME> folder.

4. Linking Dari Clustergen Voice to Flite 1.5.6

The previous section described enabling Dari (and Pashto) on Flite 1.5.6. Enabling Dari only means that Flite 1.5.6 is able to link with the Dari voice library. Flite 1.5.6 still requires the Dari voice library and cannot synthesize a Dari voice on its own. This section describes linking the Dari Clustergen voice library to the Dari-enabled Flite 1.5.6.

1. **Download Dari Clustergen.**

   Please contact Dr Alan Black[5] at CMU for instructions on downloading the Dari voice libraries.

2. **Extract Dari Clustergen to disk.**

   Once the Dari Clustergen archive is downloaded, extract the archive to disk. We extracted the Dari Clustergen to "/Flite/cmu_dari_transtac_male_cg3". For the remainder of this report, "/Flite/cmu_dari_transtac_male_cg3" is referenced as <DARI_CG3_HOME>.

3. **Configure Dari Clustergen.**

   Dari Clustergen needs to know where Flite 1.5.6 is located. Open the Makefile located at "<DARI_CG3_HOME>/flite/Makefile" with a text editor and update line 33. Replace the existing line

   ```
   FLITEDIR=/home/awb/projects/arl/flite/
   ```

   with the path to Flite 1.5.6

   ```
   FLITEDIR=/Flite/flite-1.5.6-current_w_1.9.0_Dari_Pashto/
   ```

4. **Compile the Dari Clustergen voice.**

   Use the following commands to compile the Dari Clustergen voice:

   ```
   cd <DARI_CG3_HOME>/flite/
   make clean
   make
   ```

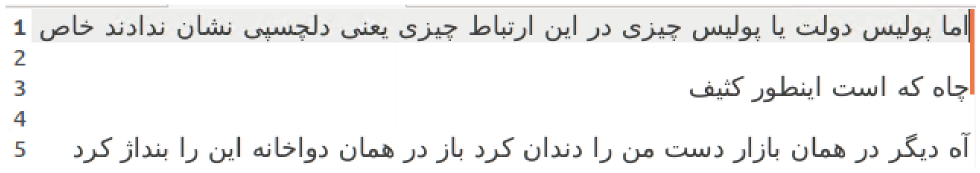
   Successful compilation creates several new files, including "flite_cmu_dari_transtac_male_cg3", under the "<DARI_CG3_HOME>/flite" folder.

5. **Verify the Dari Clustergen voice.**

   If Dari Clustergen was installed correctly, it should synthesize a Dari voice using a sample Dari text file (included with the Dari distribution). Again, OSS is required to hear the synthesized Dari voice. The sample text file command is as follows:

```
cd <DARI_CG3_HOME>
./flite/flite_cmu_dari_transtac_male_cg3 -f dari.txt
```

This test command takes several minutes to complete because the Dari sample file contains more than 160 lines of Dari text. Developers may wish to delete the majority of the lines to reduce the processing time. Figure 1 shows a Dari sample text file.



```
اما پولیس دولت یا پولیس چیزی در این ارتباط چیزی یعنی دلچسپی نشان ندادند خاص

چاه که است اینطور کثیف

آه دیگر در همان بازار دست من را دندان کرد باز در همان دواخانه این را بنداژ کرد
```

Fig. 1 Dari sample text file included in the Dari Clustergen distribution

5. Android Development Tool (ADT) and Android Native Development Kit (NDK) Setup

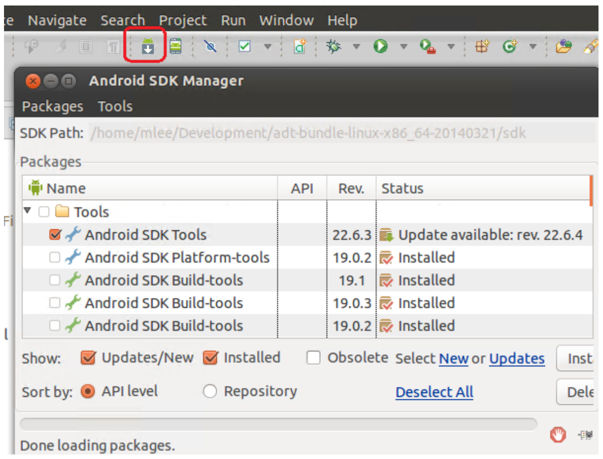
Before any Android project can be compiled, an Android development environment needs to be set up. There are several ways to compile an Android app. We chose to develop our app using the Android Development Tool (ADT) and Eclipse. Both are packaged as a single download and are available from the Android developer Website.[7] This report assumes that both ADT and Eclipse are used as the Android Integrated Development Environment and are installed correctly to compile an Android project. In addition to ADT and Eclipse, the Android Native Development Kit (NDK) is required to invoke C/C++ methods used in the Android Flite Engine. The Android NDK is also available from the Android developer Website.[8] ADT and NDK setup instructions and troubleshooting help are widely available so this report does not describe them. For the remainder of this report, we reference "<ADT_HOME>" as where ADT was extracted, and "<NDK_HOME>" as where NDK was extracted.

After ADT and NDK are set up, one must add the following environment variables to the system:

```
export FLITEDIR=/Flite/flite-1.5.6-current_w_1.9.0_Dari_Pashto
export FLITE_APP_DIR=/home/arl/workspace/FliteEngine
export ANDROID_NDK=/Development/android-ndk-r9d
export ANDROID_SDK=/Development/adt-bundle-linux-x86_64-20140321/sdk
```

The environment variable "FLITE_APP_DIR" is the path to the Android Flite Engine application. Up until this point, we have not discussed the Android Flite Engine application in detail. For now, one should create an environment variable that points to the folder where the Android Flite Engine project will be saved. This is shown in the second command listed above.

As part of the ADT download, the Eclipse Integrated Development Environment (IDE) is downloaded as well. The Eclipse launcher is located at <ADT_HOME>/eclipse/eclipse. We chose this included version of Eclipse to develop the Android Flite Engine, because it is already preconfigured for Android development. Please verify Eclipse is able to launch without any errors, and use the built-in Android SDK Manager (Fig. 2) to download recent versions of Android SDK tools before continuing.


e Navigate Search Project Run Window Help
Android SDK Manager
Packages Tools
Show: Updates/New Installed Obsolete Select New or Updates
Inst
Sort by: API level Repository
Deselect All
Dele

SDK Path: /home/mlee/Development/adt-bundle-linux-x86_64-20140321/sdk

Packages

Name	API	Rev.	Status
Tools			
✓ Android SDK Tools		22.6.3	Update available: rev. 22.6.4
Android SDK Platform-tools		19.0.2	Installed
Android SDK Build-tools		19.1	Installed
Android SDK Build-tools		19.0.3	Installed
Android SDK Build-tools		19.0.2	Installed

Show: [✓ Updates/New] [✓ Installed] Obsolete

Sort by: [✓ API level] Repository

Done loading packages.

Fig. 2 Android SDK Manager included in Eclipse

6. Compiling Dari Clustergen Library for Multiple CPU Architectures

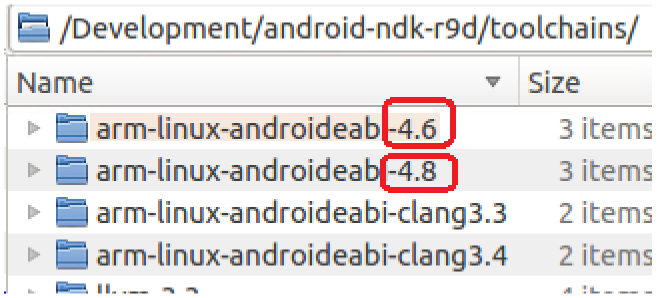
Most Android devices run on ARM processors instead of x86 processors. Typically, a developer is not concerned about this aspect when creating an Android app, because the Android compiler will automatically build for multiple target processors and shelter developers from this process. However, when an Android app imports and invokes an external native library, this library must be compiled for multiple central processing unit (CPU) architectures. Fortunately, Dr Black and his team have designed their *Makefile* so that it can compile for various CPU architectures. This part of the migration effort is the most lengthy and repetitive process, because it requires several steps for each CPU type and we need to compile for multiple architectures. In this section, we describe step-by-step instructions on how to compile the Dari Clustergen library for armeabi, armeabiv7a, x86, and Mips. The key information here is that the CPU target for Flite 1.5.6 is also the target to use when compiling the Dari Clustergen voice (i.e., Flite 1.5.6 must be compiled first and then Dari Clustergen compiled immediately afterwards).

1. Miscellaneous pre-conditions

Flite 1.5.6 was shipped pre-configured to compile on 32-bit Linux using Android GCC, version 4.4.3. If the developer is not using the same combination, compilation will throw the following "Command not found" exception:

```
make[2]: /Development/android-ndk-r9d/toolchains/arm-linux-androideabi-
4.4.3/prebuilt/linux-x86/bin/arm-linux-androideabi-gcc: Command not found
make[2]: *** [../../build/armeabi-android/obj/src/audio/auclient.o] Error
127
make[1]: *** [../build/armeabi-android/obj/src/.make_build_dirs] Error 2
make: *** [build/armeabi-android/obj//.make_build_dirs] Error 2
```

We developed on 64-bit Linux and Android GCC, version 4.8, so we had to update <FLITE_HOME>/configure.in with the correct paths before compiling the Dari Clustergen for other CPU architectures. The collection of installed Android C Compilers are viewable by listing files in the "$ANDROID_NDK/toolchains" folder (Fig. 3). Android C Compiler versions are indicated by the suffix of subdirectory names. For example, our Android NDK contained Android C Compiler versions 4.6 and 4.8.


/Development/android-ndk-r9d/toolchains/

Name	Size
arm-linux-androideabi-4.6	3 items
arm-linux-androideabi-4.8	3 items
arm-linux-androideabi-clang3.3	2 items
arm-linux-androideabi-clang3.4	2 items

Fig. 3 Android C Compiler versions listed in $ANDROID_NDK/toolchains

Make the following three changes to <FLITE_HOME>/configure.in.

At line 157 (actual line number may vary), replace the line

```
ANDROID_GCC_VERSION=4.4.3
```

with

```
ANDROID_GCC_VERSION=4.8
```

At line 180 (actual line number may vary), replace the line

```
ANDROIDBIN="$ANDROID_NDK/toolchains/x86-
$ANDROID_GCC_VERSION/prebuilt/linux-x86_64/bin/i686-android-linux"
```

with

```
ANDROIDBIN="$ANDROID_NDK/toolchains/x86-
$ANDROID_GCC_VERSION/prebuilt/linux-x86_64/bin/i686-linux-android"
```

Globally replace the string

```
linux-x86
```

with

```
linux-x86_64
```

There should be four occurrences.

The following is an example of the line that was updated:

```
ANDROIDBIN="$ANDROID_NDK/toolchains/arm-linux-androideabi-
$ANDROID_GCC_VERSION/prebuilt/linux-x86_64/bin/arm-linux-androideabi"
```

De-reference the environment variables and make sure that the referenced paths exist on disk.

After <FLITE_HOME>/configure.in is updated, Flite must be reconfigured using the following command:

```
cd <FLITE_HOME>
autoreconf
```

Developers working with Flite 1.5.6 and Dari Clustergen can skip to the next step. But developers using Flite 1.9.0 or Dari Unit Select must make additional changes.

For developers migrating Dari Unit Select, the following update is required to compile:

Edit `<cmu_dari_transtac_male_cl12t>/flite_g721vuv/`**`Makefile`**

An environment variable called "ANDROID_NDK_SYSROOT" is used in this edit. A developer does not need to explicitly add this environment variable to the system; it is added automatically as part of the build process by Flite.

At line 100 (actual line number may vary), change the lines

```
$(VOICENAME)_lpc.o: $(VOICENAME)_lpc.c
     $(CC) -I. -I$(FLITEDIR)/include -c -o $@ $<
```

to

```
$(VOICENAME)_lpc.o: $(VOICENAME)_lpc.c
     $(CC) -I. -I$(FLITEDIR)/include –I$(ANDROID_NDK_SYSROOT)/usr/include -c -o $@ $<
```

On the next line down, change the lines

```
$(VOICENAME)_mcep.o: $(VOICENAME)_mcep.c
     $(CC) -I. -I$(FLITEDIR)/include -c -o $@ $<
```

to

```
$(VOICENAME)_mcep.o: $(VOICENAME)_mcep.c
     $(CC) -I. -I$(FLITEDIR)/include –I$(ANDROID_NDK_SYSROOT)/usr/include -c -o $@ $<
```

Unless these library paths are included, the following exception will be thrown.

```
/Development/android-ndk-r9d/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-
x86_64/lib/gcc/arm-linux-androideabi/4.8/include-fixed/stdio.h:50:23: fatal error:
sys/cdefs.h: No such file or directory
 #include <sys/cdefs.h>
                       ^
Compilation terminated.
```

For developers using *Dari-enabled Flite Engine 1.9.0* (instead of Flite 1.5.6), the following two updates are required in "<FLITE_HOME>/configure.sub" to compile Flite for Android platforms (e.g., armeabiv7a).

Edit #1

Locate the text at line 256 (actual line number may vary):

```
| arm | arm[bl]e | arme[lb] | armv[2-8] | armv[3-8][lb] |
armv7[arm] \
```

Add a new line at the end and paste the following text:

```
| arc | arm | armeabi | armeabiv7a | arm[bl]e | arme[lb] | armv[2345] |
armv[345][lb] | avr \
```

Edit #2

Locate the entry at line 1422 (actual line number may vary):

```
-wince*)
  os=-wince
  ;;
```

Add a new line at the end and append the following entry:

```
-android*)
  os=-android
  ;;
```

Unless the above two edits are made in "<FLITE_HOME>/configure.sub", a developer may encounter the following compilation exception:

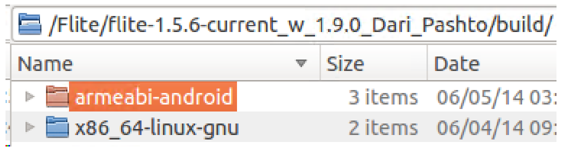
```
checking target system type... Invalid configuration `armeabiv7a-android':
machine `armeabiv7a' not recognized
configure: error: /bin/bash ./config.sub armeabiv7a-android failed
```

2. Dari Clustergen for "armeabi-android"

Compile Flite for the armeabi platform:

```
cd <FLITE_HOME>
./configure --with-langvox=transtac --target=armeabi-android
make clean
make
```

The above command creates a new directory called "<FLITE_HOME>/build/armeabi-android/" and appropriate subdirectories under it (Fig. 4).



Name	Size	Date
armeabi-android	3 items	06/05/14 03:
x86_64-linux-gnu	2 items	06/04/14 09:

Fig. 4 The "armeabi-android" folder created after compiling Flite for armeabi platform

Compile Dari Clustergen for the armeabi platform:

```
cd <DARI_CG3_HOME>/flite
make clean
make
```

Verify that Dari Clustergen library, libcmu_dari_transtac_male_cg3.a, was compiled for ARM:

```
 cd <DARI_CG3_HOME>/flite
 cp libcmu_dari_transtac_male_cg3.a test/
 cd test/
 ar -x libcmu_dari_transtac_male_cg3.a
 file cmu_dari_transtac_male_cg3.o
 cmu_dari_transtac_male_cg3.o: ELF 32-bit LSB  relocatable, ARM, EABI5
version 1 (SYSV), not stripped
```

Copy the new libcmu_dari_transtac_male_cg3.a to Flite 1.5.6.:

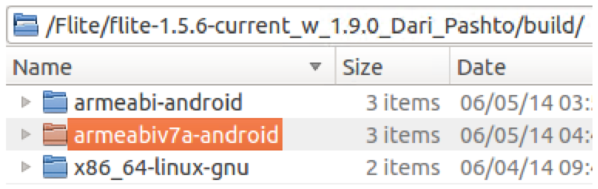
```
cp <DARI_CG3_HOME>/flite/libcmu_dari_transtac_male_cg3.a
<FLITE_HOME>/build/armeabi-android/lib
```

3. Dari Clustergen for "armeabiv7a-android"

Compile Flite for the armeabiv7a platform:

```
cd <FLITE_HOME>
./configure --with-langvox=transtac --target=armeabiv7a-android
make clean
make
```

The above command will create a new directory named "<FLITE_HOME>/build/armeabiv7a-android/" and subdirectories under it (Fig. 5).



Name	Size	Date
armeabi-android	3 items	06/05/14 03:
armeabiv7a-android	3 items	06/05/14 04:
x86_64-linux-gnu	2 items	06/04/14 09:

Fig. 1 The "armeabiv7a-android" folder created after compiling Flite for armeabiv7a platform

Compile Dari Clustergen for the armeabiv7a platform:

```
cd <DARI_CG3_HOME>/flite
make clean
make
```

Verify that Dari Clustergen library, libcmu_dari_transtac_male_cg3.a, was compiled for ARM:

```
cd <DARI_CG3_HOME>/flite
cp libcmu_dari_transtac_male_cg3.a test/
cd test/
ar -x libcmu_dari_transtac_male_cg3.a
file cmu_dari_transtac_male_cg3.o
cmu_dari_transtac_male_cg3.o: ELF 32-bit LSB  relocatable, ARM, EABI5
version 1 (SYSV), not stripped
```

Copy the new libcmu_dari_transtac_male_cg3.a to Flite 1.5.6.:

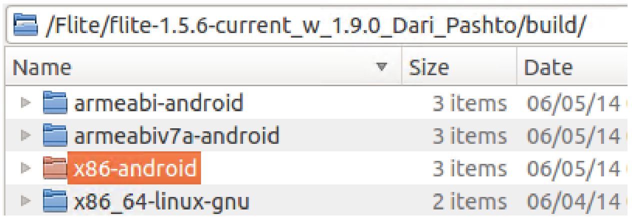
```
cp <DARI_CG3_HOME>/flite/libcmu_dari_transtac_male_cg3.a
<FLITE_HOME>/build/armeabiv7a-android/lib
```

4. Dari Clustergen for "x86-android"

Compile Flite for the x86 platform:

```
cd <FLITE_HOME>
./configure --with-langvox=transtac --target=x86-android
make clean
make
```

The above command should have created new directory "<FLITE_HOME>/build/x86-android/" and subdirectories under it (Fig. 6).



Name	Size	Date
armeabi-android	3 items	06/05/14
armeabiv7a-android	3 items	06/05/14
x86-android	3 items	06/05/14
x86_64-linux-gnu	2 items	06/04/14

Fig. 6 The "x86-android" folder created after compiling Flite for the x86 platform

Compile Dari Clustergen for the x86 platform:

```
cd <DARI_CG3_HOME>/flite
make clean
make
```

Verify that Dari Clustergen library, libcmu_dari_transtac_male_cg3.a, was compiled for x86:

```
cd <DARI_CG3_HOME>/flite
cp libcmu_dari_transtac_male_cg3.a test/
cd test/
ar -x libcmu_dari_transtac_male_cg3.a
file cmu_dari_transtac_male_cg3.o
cmu_dari_transtac_male_cg3.o: ELF 32-bit LSB  relocatable, Intel 80386,
version 1 (SYSV), not stripped
```

Copy the new libcmu_dari_transtac_male_cg3.a to Flite 1.5.6.:

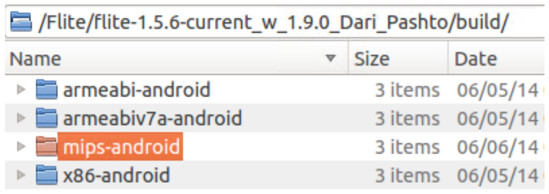
```
cp <DARI_CG3_HOME>/flite/libcmu_dari_transtac_male_cg3.a
<FLITE_HOME>/build/x86-android /lib
```

5. Dari Clustergen for "mips-android"

Compile Flite for the mips platform:

```
cd <FLITE_HOME>
./configure --with-langvox=transtac --target=mips-android
make clean
make
```

The above command will create a new directory called "<FLITE_HOME>/build/mips-android/" and subdirectories under it (Fig. 7).



/Flite/flite-1.5.6-current_w_1.9.0_Dari_Pashto/build/

Name	Size	Date
armeabi-android	3 items	06/05/14
armeabiv7a-android	3 items	06/05/14
mips-android	3 items	06/06/14
x86-android	3 items	06/05/14

Fig. 7 The "mips-android" folder created after compiling Flite for the mips platform

Compile Dari Clustergen for the mips platform:

```
cd <DARI_CG3_HOME>/flite
make clean
make
```

Verify Dari Clustergen library, libcmu_dari_transtac_male_cg3.a, was compiled for mips:

```
cd <DARI_CG3_HOME>/flite
cp libcmu_dari_transtac_male_cg3.a test/
cd test/
ar -x libcmu_dari_transtac_male_cg3.a
file cmu_dari_transtac_male_cg3.o
cmu_dari_transtac_male_cg3.o: ELF 32-bit LSB  relocatable, MIPS, MIPS32
version 1 (SYSV), not stripped
```

Copy the new libcmu_dari_transtac_male_cg3.a to Flite 1.5.6.:

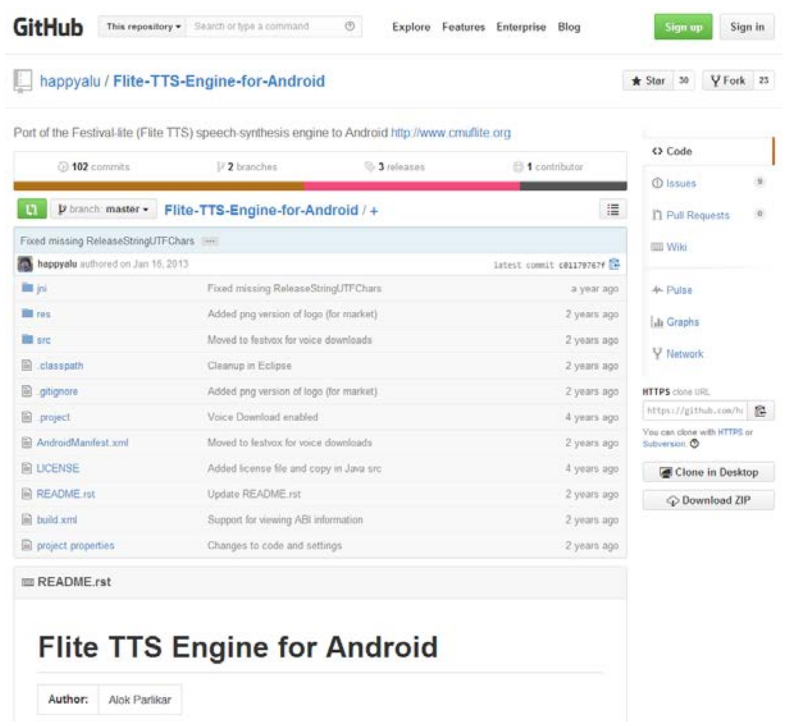
```
cp <DARI_CG3_HOME>/flite/libcmu_dari_transtac_male_cg3.a
<FLITE_HOME>/build/mips-android /lib
```

At this point, the Dari Clustergen library has been compiled for all CPU platforms and installed in Flite. All the library compilations required outside of Eclipse are now complete, and we are ready to set up the Android Flite Engine and then integrate Dari Clustergen to the Android Flite Engine.

7. Flite Android Engine Setup

As mentioned, the Flite Android Engine is a small Android project developed by Dr Parlikar that wraps Flite with Google's TTS layer and allows developers to invoke Flite voices in Android. It also features a user interface that lets a user download Flite voices, select Flite voices, and synthesize a voice from typed text. To be clear, the Flite Android Engine is not a re-implementation of Flite on the Android platform. The engine still requires the standalone Flite as a dependency. Currently, there is no easy way to use Flite Android Engine's user interface to import the Dari Clustergen voice to the Flite Android Engine. The Dari voice must be integrated in the source code.

Before we can integrate Dari Clustergen to the Flite Android Engine, we need to download, compile, and verify that the default version works out-of-the-box. The Flite Android Engine project page, moderated by Dr Parlikar, is found at a public repository called GitHub[9] (Fig. 8). Follow the instructions from the project Webpage to download and set up the Android Flite Engine. The project Webpage describes using Apache Another Neat Tool (ANT) to build the project, but we use Eclipse to import and build the project. When the archive file is imported to Eclipse, an Android project called "FliteEngine" is automatically created (Fig. 9).



happyalu / Flite-TTS-Engine-for-Android

Port of the Festival-lite (Flite TTS) speech-synthesis engine to Android http://www.cmuflite.org

102 commits · 2 branches · 3 releases · 1 contributor

branch: master · Flite-TTS-Engine-for-Android / +

Fixed missing ReleaseStringUTFChars

happyalu authored on Jan 16, 2013 · latest commit c01179767f

jni	Fixed missing ReleaseStringUTFChars	a year ago
res	Added png version of logo (for market)	2 years ago
src	Moved to festvox for voice downloads	2 years ago
.classpath	Cleanup in Eclipse	2 years ago
.gitignore	Added png version of logo (for market)	2 years ago
.project	Voice Download enabled	4 years ago
AndroidManifest.xml	Moved to festvox for voice downloads	2 years ago
LICENSE	Added license file and copy in Java src	4 years ago
README.rst	Update README.rst	2 years ago
build.xml	Support for viewing ABI information	2 years ago
project.properties	Changes to code and settings	2 years ago

HTTPS clone URL https://github.com/h

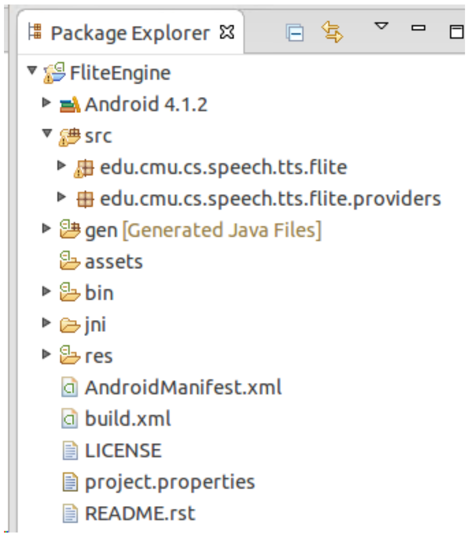
You can clone with HTTPS or Subversion.

README.rst

Flite TTS Engine for Android

Author: Alok Parlikar

Fig. 8 Flite Android Engine project available from GitHub



```
Package Explorer
▼ FliteEngine
  ▶ Android 4.1.2
  ▼ src
    ▶ edu.cmu.cs.speech.tts.flite
    ▶ edu.cmu.cs.speech.tts.flite.providers
  ▶ gen [Generated Java Files]
    assets
  ▶ bin
  ▶ jni
  ▶ res
    AndroidManifest.xml
    build.xml
    LICENSE
    project.properties
    README.rst
```

Fig. 9 Android Flite Engine project imported to Eclipse

As soon as the project is imported, the Flite Android Engine is able to launch its main graphical user interface (GUI) without any configuration. But its features will not function because it has not been linked to Flite. One can verify the Flite Android Engine project was imported to Eclipse correctly by launching the app to a device. Successful launch displays the following menu on the device (Fig. 10). However, one should not attempt to access any features on the app as it will throw an error because the app is not yet configured with Flite.

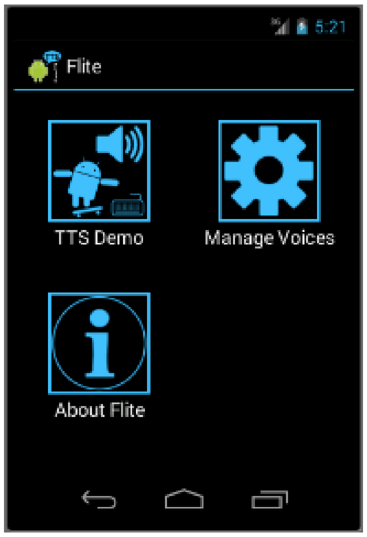


Fig. 10 Main page of the Android Flite Engine

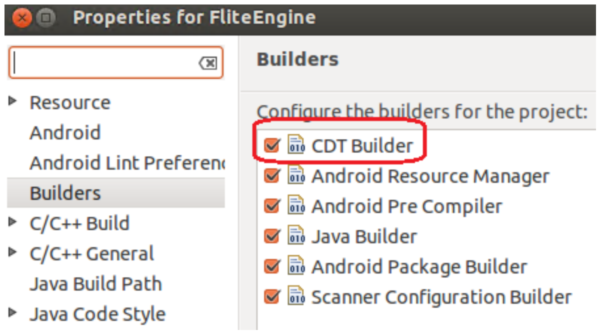
Once the project import is verified, we need to configure the Flite Android Engine and link it to the main Flite installation. This is indirectly accomplished by generating Flite shared library file (e.g., libttsfile.so) for each of the CPU targets in the Flite Android Engine project. It is very important to generate the libttsfile.so shared library, because it is the very heart of the Flite Android Engine app and it is also what performs the voice synthesis task. The Dari Clustergen voices created and saved in <FLITE_HOME> from the previous section are also included in this libttsfile.so shared library. There are several ways to generate the shared library. The easiest method is by allowing Eclipse to generate it. The only non-intuitive step is initially configuring Eclipse to generate libttsfile.so as part of the build process. Although the Flite Android Engine project will build and run without generating the shared library, the app will crash and throw the following error once the TTS function is invoked:

```
java.lang.UnsatisfiedLinkError: Couldn't load ttsflite: findLibrary
returned null
```

Follow these instructions to enable "CDT Builder" (C/C++ Development Tooling) in the project properties section and build the libttsfile.so shared library.

1. **Verify "CDT Builder" is available and enabled.**

   View the list of available builders in the "Builders" section in the project properties page. There should be a check box for "CDT Builder" (Fig. 11). If "CDT Builder" is not listed, then proceed to the next step. If "CDT Builder" is listed, then skip the next step.



Builders

Configure the builders for the project:

- [✓] CDT Builder
- [✓] Android Resource Manager
- [✓] Android Pre Compiler
- [✓] Java Builder
- [✓] Android Package Builder
- [✓] Scanner Configuration Builder

Fig. 11 "CDT Builder" listed as an available option in the FliteEngine properties page

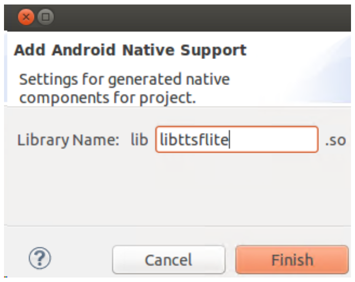
2. **Enable Android Native Support.**

   Open the "Add Android Native Support" page accessible under the following:

```
FliteEngine properties -> Android Tools -> Add Android Native Support
```

   Enter "libttsflite" as the library name and click "Finish".

   "CDT Builder" should now be included with the available builders in the "Builders" section in the project properties page (Fig. 12.).



Add Android Native Support

Settings for generated native components for project.

Library Name: lib libttsflite .so

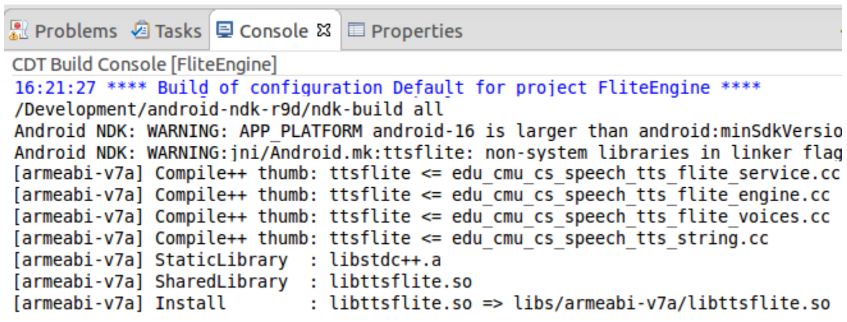
Fig. 12 Adding Android Native Support to the FliteEngine project

3. **Build the project.**

   The build command is accessible under the following:

```
Project -> Build Project
```

   This command now invokes `<NDK_HOME>/ndk-build` as part of the project's build process and generates libttsfile.so for each CPU target: armeabi, armeabi-v7a, mips, and x86 (Fig. 13).



```
16:21:27 **** Build of configuration Default for project FliteEngine ****
/Development/android-ndk-r9d/ndk-build all
Android NDK: WARNING: APP_PLATFORM android-16 is larger than android:minSdkVersio
Android NDK: WARNING:jni/Android.mk:ttsflite: non-system libraries in linker flag
[armeabi-v7a] Compile++ thumb: ttsflite <= edu_cmu_cs_speech_tts_flite_service.cc
[armeabi-v7a] Compile++ thumb: ttsflite <= edu_cmu_cs_speech_tts_flite_engine.cc
[armeabi-v7a] Compile++ thumb: ttsflite <= edu_cmu_cs_speech_tts_flite_voices.cc
[armeabi-v7a] Compile++ thumb: ttsflite <= edu_cmu_cs_speech_tts_string.cc
[armeabi-v7a] StaticLibrary  : libstdc++.a
[armeabi-v7a] SharedLibrary  : libttsflite.so
[armeabi-v7a] Install        : libttsflite.so => libs/armeabi-v7a/libttsflite.so
```

Fig. 13   Building "libttsflite.so" in Eclipse using NDK

4. **Verify generated shared libraries.**

   If the libttsfile.so libraries are generated correctly, they should exist in the "libs" folder under their corresponding CPU target (Fig. 14).

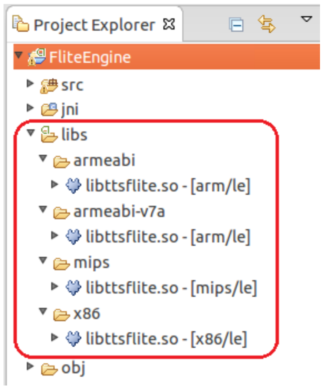
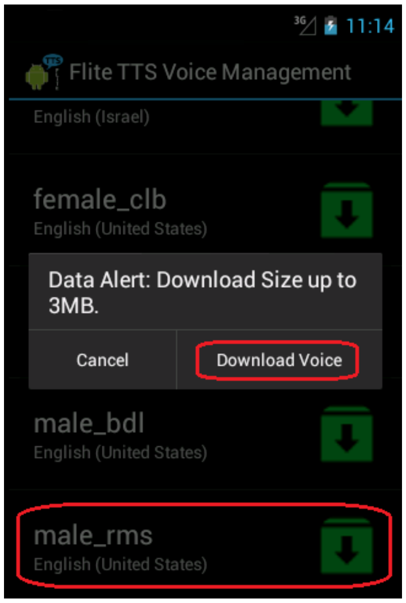

Project Explorer
FliteEngine
src
jni
libs
armeabi
libttsflite.so - [arm/le]
armeabi-v7a
libttsflite.so - [arm/le]
mips
libttsflite.so - [mips/le]
x86
libttsflite.so - [x86/le]
obj

Fig. 14   The "libttsflite.so" generated in the FliteEngine project

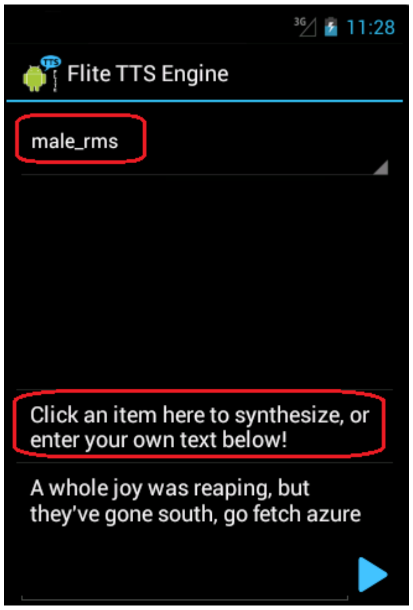
The Flite Android Engine is now set up and ready for use. Out-of-the-box, the Flite Android Engine does not contain any voices. Each voice must be downloaded separately. To do this, from the main menu, click on the "Manage Voices" icon to see a list of downloadable voices. As a test, click on the "male_rms" icon to download this English voice to the device (Fig. 15).



Data Alert: Download Size up to 3MB.

Fig. 15 The “Flite TTS Voice Management” page. Download a “male_rms” English voice.

Once the “male_rms” voice is downloaded, go back to the main menu and click on the “TTS Demo” icon. The demo page will be pre-configured for the English “male_rms” voice, because “male_rms” was the only voice downloaded. Click on a sample English text to hear the text synthesized in an English voice (Fig. 16).



Click an item here to synthesize, or enter your own text below!

A whole joy was reaping, but they've gone south, go fetch azure

Fig. 16 The “TTS Demo” page with a “male_rms” English voice

If there are any errors that occur during this test or certain functions cause an error, environment variables may be the cause. Verify that the required environment variables (e.g., FLITEDIR, FLITE_APP_DIR, ANDROID_NDK, and ANDROID_SDK) are defined in the development environment. An alternative method is to define these variables directly in the Android.mk configuration file in the Flite Android Engine project. Add the following lines at the beginning of the file:

```
FLITEDIR:=/Flite/flite-1.5.6-current_w_1.9.0_Dari_Pashto
FLITE_APP_DIR:=/workspace/FliteEngine
ANDROID_NDK:=/Development/android-ndk-r9d
ANDROID_SDK:=/Development/adt-bundle-linux-x86_64-20140321/sdk
```

8. Integrating Dari Clustergen Library into the Flite Android Engine Source Code

At this point, the Dari Clustergen voices are compiled, the libttsflite.so shared libraries are generated, the Flite Android Engine compiles on Eclipse, and English voices are synthesized on an Android platform. The majority of the developers interested in migrating Dari Clustergen to an Android platform will find this section the most pertinent in the entire report.

We acquired the instructions to add and link the Dari Clustergen voice from numerous correspondences with Dr Parlikar, and these are described in step 1 and 2. Step 3 is a quick "hack" we implemented to configure Flite for Dari Clustergen. This "hack" was due to the difficulty creating a valid ISO 639 compliant Java Locale object for Dari. Other developers integrating common widely used languages may be able to skip step 3.

Although we described integrating Dari Clustergen, these instructions can be used for any Clustergen voices. Again, Dr Parlikar stated that these instructions are also applicable for Dari Unit Select voices, but we were not able to synthesize Dari text using Dari Unit Select.

Now we describe steps to integrate Dari Clustergen library in the Android Flite Engine source code. This portion primarily involves editing source code in Eclipse. We describe changes needed in each file by showing before and after edit.

1. **Edit `<FLITE_APP_DIR>/jni/Android.mk`**

   Edit #1 – Update the definition of "LOCAL_LDLIBS" at line 74 (actual line number may vary).

   This edit tells the Flite Android Engine project where the Dari Clustergen libraries are physically located.

   "$(FLITE_LIB_DIR)" refers to the folder where we saved a copy of Dari Clustergen voice for each CPU target (e.g., "<FLITE_HOME>/build/armeabiv7a-android/lib").

Before edit:

```
LOCAL_LDLIBS:= -llog \
        $(FLITE_LIB_DIR)/libflite_cmulex.a \
        $(FLITE_LIB_DIR)/libflite_usenglish.a \
        $(FLITE_LIB_DIR)/libflite.a \
```

After edit:

```
LOCAL_LDLIBS:= -llog \
        $(FLITE_LIB_DIR)/libcmu_dari_transtac_male_cg3.a \
        $(FLITE_LIB_DIR)/libflite_cmulex.a \
        $(FLITE_LIB_DIR)/libflite_usenglish.a \
        $(FLITE_LIB_DIR)/libflite_cmu_dari_lex.a \
        $(FLITE_LIB_DIR)/libflite_cmu_dari_lang.a \
        $(FLITE_LIB_DIR)/libflite.a \
```

2. **Edit `<FLITE_APP_DIR>/jni/edu_cmu_cs_speech_tts_flite_engine.cc`**

Edit #1 – Add Dari language support at line 68 (actual line number may vary) of edu_cmu_cs_speech_tts_flite_engine.cc.

In order to add Dari language support, we need to look up the correct method names. Open `<FLITE_HOME>/main/flite_lang_list.c` and note the name of the Dari initialization method. We noted the following method names from `<FLITE_HOME>/main/flite_lang_list.c`. The highlighted method names are used in this step:

```
. . .
void cmu_dari_lang_init(cst_voice *v);
cst_lexicon *cmu_dari_lex_init(void);
. . .
void flite_set_lang_list(void)
{
   flite_add_lang("utf8_grapheme",utf8_grapheme_lang_init,utf8_grapheme_lex_init);
   flite_add_lang("usenglish",usenglish_init,cmulex_init);
   flite_add_lang("cmu_dari_lang",cmu_dari_lang_init,cmu_dari_lex_init);
. . .
}
```

Three lines were used in edu_cmu_cs_speech_tts_flite_engine.cc.

Before edit:

```
// Declarations
extern "C" void usenglish_init(cst_voice *v);
extern "C" cst_lexicon *cmulex_init(void);
. . .
flite_add_lang("eng",usenglish_init,cmulex_init);
```

After edit:

```
// Declarations
extern "C" void usenglish_init(cst_voice *v);
extern "C" cst_lexicon *cmulex_init(void);
extern "C" void cmu_dari_lang_init(cst_voice *v);
extern "C" cst_lexicon *cmu_dari_lex_init(void);
. . .
flite_add_lang("eng",usenglish_init,cmulex_init);
flite_add_lang("cmu_dari_lang", cmu_dari_lang_init, cmu_dari_lex_init);
```

Edit #2 – Enable the Dari linked voice at line 84 (actual line number may vary) of edu_cmu_cs_speech_tts_flite_engine.cc.

Before edit:

```
loadedVoices = new FliteEngine::Voices(0,
FliteEngine::ONLY_ONE_VOICE_REGISTERED);
```

After edit:

```
loadedVoices = new FliteEngine::Voices(10,
FliteEngine::ONLY_ONE_VOICE_REGISTERED);
```

The first parameter sets the maximum number of linked voices loaded to Flite. This is a maximum number. It is okay to set a high number.

The second parameter should be left as "FliteEngine::ONLY_ONE_VOICE_REGISTERED", which specifies that Flite will handle one voice at a time.

Edit #3 – Add the Dari linked voice at line 87 (actual line number may vary) of edu_cmu_cs_speech_tts_flite_engine.cc.

In order to add the Dari linked voice, we need to look up the correct method names. Open `<DARI_CG3_HOME>/flite/voxdefs.h` and note the Dari register and unregister method names. We noted the following method names from `<DARI_CG3_HOME>/flite/voxdefs.h`. The highlighted method names are used in this step:

```
#define VOXNAME cmu_dari_transtac_male_cg3
#define REGISTER_VOX register_cmu_dari_transtac_male_cg3
#define UNREGISTER_VOX unregister_cmu_dari_transtac_male_cg3
#define VOXHUMAN "transtac_male_cg3"
. . .
```

Before edit:

```
extern "C" cst_lexicon *cmu_dari_lex_init(void);
. . .

loadedVoices = new FliteEngine::Voices(10,
FliteEngine::ONLY_ONE_VOICE_REGISTERED);

if(loadedVoices == NULL)
. . .
```

After edit:

```
extern "C" cst_lexicon *cmu_dari_lex_init(void);
extern "C" cst_voice *register_cmu_dari_transtac_male_cg3(const char
*voxdir);
extern "C" void unregister_cmu_dari_transtac_male_cg3(cst_voice *vox);
```

```
. . .

loadedVoices = new FliteEngine::Voices(10,
FliteEngine::ONLY_ONE_VOICE_REGISTERED);
loadedVoices->AddLinkedVoice("cmu_dari_lang", "AFG", "dari_cg3",
register_cmu_dari_transtac_male_cg3,
unregister_cmu_dari_transtac_male_cg3);

if(loadedVoices == NULL)
. . .
```

The first three parameters of the “AddLinkedVoice” method are Language, Country, and Variant. These three values are used by the Flite Android Engine to create a Java Locale object. Technically, we are supposed to use ISO 639 compliant Language and Country values, but the values we used are not ISO 639 compliant because we are integrating Dari. This issue is discussed further in a later section. For our purpose, these parameters are sufficient. The string “dari_cg3” is the identifier for Dari Clustergen.

3. **Edit `<FLITE_APP_DIR>/src/edu/cmu/cs/speech/tts/flite/FliteTtsService.java`**

Edit #1 – Add a conditional check for “dari_cg3” in the onSynthesizeText method at line 132 (actual line number may vary).

This step can be skipped for developers who are able to create a valid Java Locale object using the integrating language. The existing code will be able to configure Flite for that language. However, Dari Clustergen requires modification because the Language and Country values we defined (from step 2) for Dari Clustergen will not generate a valid Java Locale object. Flite Android Engine will not invoke the set language method for Dari Clustergen unless we make changes to the project. Instead of making significant changes throughout the project to adjust with the current workflow, we chose to implement a simple “hack” to minimize alterations to the Flite Android Engine project. Existing code invokes a set language method using the Language, Country, and Variant values corresponding to the selected voice. Our change simply over writes the existing set language method by invoking the set language method the second time with correct values if selected voice is Dari Clustergen. Developers may wish to implement a more elegant approach to avoid hard-coding language and country values. The important take away is that the set language method must be invoked with the values previously defined for Dari.

Before edit:

```
if (! ((mLanguage == language) &&
        (mCountry == country) &&
        (mVariant == variant ))) {
        result = mEngine.setLanguage(language, country, variant);
. . .
}

if (!result) {
        Log.e(LOG_TAG, "Could not set language for synthesis");
```

After edit:

```
if (! ((mLanguage == language) &&
      (mCountry == country) &&
      (mVariant == variant ))) {
      result = mEngine.setLanguage(language, country, variant);
. . .
}

if(variant.equals("dari_cg3"))
      result = mEngine.setLanguage("cmu_dari_lang", "AFG", "dari_cg3");

if (!result) {
      Log.e(LOG_TAG, "Could not set language for synthesis");
```

The Dari Clustergen is now integrated into the backend of the Flite Android Engine, and the engine now knows how to synthesize Dari voice using the Dari Clustergen library. Even with the changes made in this section, a user does not yet have the ability to select Dari Clustergen among list of other available voices in the Flite Android Engine user interface. The next section describes how we added Dari as a selectable voice to the Flite Android Engine without updating the drop down list of languages in the source code.

4. **Edit `<FLITE_APP_DIR>/src/edu/cmu/cs/speech/tts/flite/TTSDemo.java` (OPTIONAL)**

   Edit #1 – Add a sample Dari text in the “buildUI” method at line 143 (actual line number may vary) in TTSDemo.java.

   This update is completely optional and is not required to integrate Dari Clustergen to the Android platform. It may be difficult to test the Dari voice synthesis if the developer does not know how to type Dari. This optional step helps non-Dari speaker test Dari voice synthesis. We found this addition very helpful during testing and debugging of the Dari integration. Instead of typing Dari text (assuming the developer knows how to type Dari) each time we wanted to test Dari voice synthesis, it is much quicker to click on a pre-populated sample Dari text.

   For developers who do not know how to type Dari, there is a sample Dari text file in `<DARI_CG3_HOME>/dari.txt`. Open the file and copy a line of Dari text. We copied the following line from the file:

```
ما به شما کمک میکنیم تا بتوانید یک شهروند خبرنگار باشید، زیرا
جامعه از شماست، انتخابات از شماست و سخنان شما برای جامعه مهم است.
```

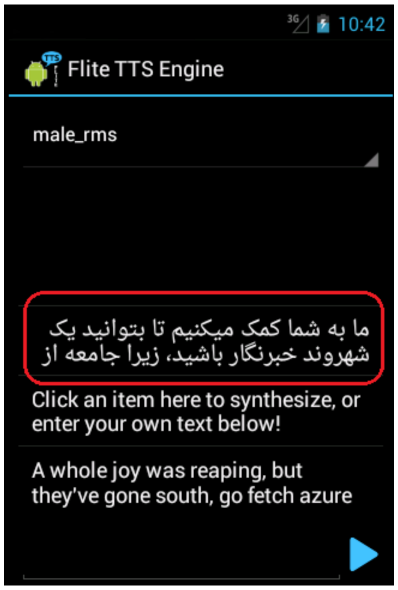
Before edit:

```
 setContentView(R.layout.activity_tts_demo);
 mStrings.add("Click an item here to synthesize, or enter your own text
below!");
```

After edit:

```
setContentView(R.layout.activity_tts_demo);
mStrings.add("ما به شما کمک میکنیم تا بتوانید یک شهروند خبرنگار باشید، زیرا جامعه
از شماست، انتخابات از شماست و سخنان شما برای جامعه مهم است.");
mStrings.add("Click an item here to synthesize, or enter your own text
below!");
```

This update adds the Dari text in the list of pre-populated texts for quick testing (Fig. 17). Note that the Flite Android Engine does not yet have the ability to set language for Dari (i.e., the Flite Android Engine will not speak Dari even if a user clicks on this Dari text). That feature is added in the next section.



ما به شما کمک میکنیم تا بتوانید یک
شهروند خبرنگار باشید، زیرا جامعه از

Click an item here to synthesize, or enter your own text below!

A whole joy was reaping, but they've gone south, go fetch azure

Fig. 17 Sample Dari text added to pre-populated list of text for quick testing

9. Adding Dari Clustergen as a Selectable Voice in the Flite Android Engine Without Coding

At this stage in the process, the Android device recognizes the Flite Android Engine as a selectable TTS Engine (e.g., Google TTS Engine) in the device settings and the Flite Android Engine now knows how to synthesize a Dari voice. However, the Flite Android Engine does not know that Dari is a selectable voice, because this voice is not available for download like other

English voices. The Dari Clustergen voice does not have a corresponding Flitevox data file, which is the resource needed to synthesize a particular voice. The Flite Android Engine assumes that each voice is equipped with a Flitevox data file, and it is downloaded to the Android device when a user chooses to enable that voice. For example, when a user chooses to enable the “male_rms” English voice, a 2.83-MB file called “male_rms.cg.flitevox” is downloaded to the “/sdcard/flite-data/cg/eng/USA” folder (Fig. 18). To add Dari as a selectable voice, Flite Android Engine must be modified to address the absence of a Dari Flitevox data file.

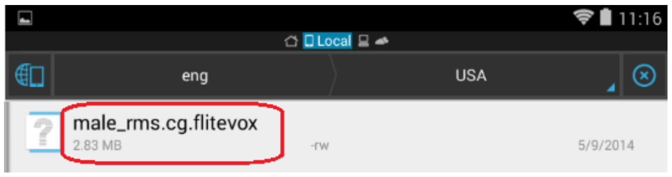
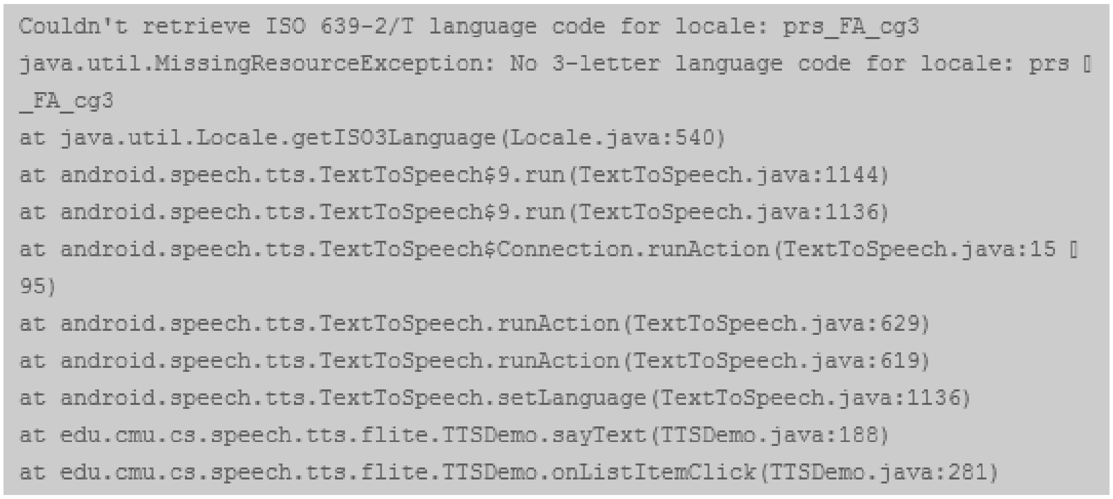


Fig. 18 “male_rms” flitevox file downloaded to “/sdcard/flite-data/cg/eng/USA”

Initially, we attempted to modify the Flite Engine source code to add the Dari voice, essentially making Dari Clustergen a permanent feature of the Flite Android Engine. This approach seemed like an easy straightforward update—add Dari Clustergen to the list of available voices and add conditions to ignore the missing Dari Flitevox data file. But we did not anticipate how extensive the Flitevox data file was verified and referenced throughout the project. The Flite Android Engine verifies the integrity of each Flitevox file by comparing its MD5 checksum against the expected value, and it is referenced in many places throughout the code. We updated the code to ignore the MD5 check for Dari and started adding several conditions to specifically ignore the missing Dari Flitevox file. This approach quickly became difficult to manage with numerous conditional checks spread throughout the project. It also would have likely caused bugs when changes were required later. In addition, it was not possible to create an exact ISO 639-2 compliant Dari Locale object, which is needed to invoke any TTS Engine on Android. Several Language and Country combinations vaguely similar to Dari were attempted (e.g., Language = “prs” + Country = “fa”, Language = “per” + Country = “fa”), but they threw a “MissingResource” exception (Fig. 19). In order to bypass the exception and continue testing, we created a Locale using Language = “ar” and Country = “DZ” even though these were not equivalent to Dari. Due to manageability concerns, we eventually decided to abandon this approach.



```
Couldn't retrieve ISO 639-2/T language code for locale: prs_FA_cg3
java.util.MissingResourceException: No 3-letter language code for locale: prs ▯
_FA_cg3
at java.util.Locale.getISO3Language(Locale.java:540)
at android.speech.tts.TextToSpeech$9.run(TextToSpeech.java:1144)
at android.speech.tts.TextToSpeech$9.run(TextToSpeech.java:1136)
at android.speech.tts.TextToSpeech$Connection.runAction(TextToSpeech.java:15 ▯
95)
at android.speech.tts.TextToSpeech.runAction(TextToSpeech.java:629)
at android.speech.tts.TextToSpeech.runAction(TextToSpeech.java:619)
at android.speech.tts.TextToSpeech.setLanguage(TextToSpeech.java:1136)
at edu.cmu.cs.speech.tts.flite.TTSDemo.sayText(TTSDemo.java:188)
at edu.cmu.cs.speech.tts.flite.TTSDemo.onListItemClick(TTSDemo.java:281)
```

Fig. 19 MissingResourceException while attempting to create a Locale object

The integration solution we chose was a much simpler workaround that only required two lines of additional code in the Android Flite Engine project. We created a MD5-valid, but non-functioning, Flitevox placeholder for the Dari voice so that the Flite Android Engine lists and accepts Dari as a valid request. The Flite Android Engine will think that it is a valid Flitevox file. We allow the Flite Android Engine to validate the Dari voice request using the surrogate Flitevox file. After the validation checks are complete, we divert the request to the real Dari Clustergen linked library instead of the surrogate Flitevox file. With this approach, adding future voices is very manageable, requiring changes to a single line of code. However, this solution requires some file manipulations and editing on the device. This section describes the workaround to adding Dari Clustergen voice to the Flite Android Engine.

1. Start the Flite App.
2. Go to the "Mange Voices" page.
3. Download any voice (if not already downloaded).

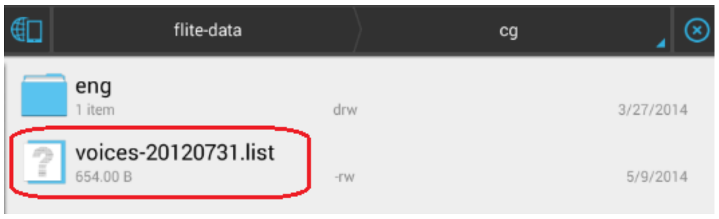
   For this example, we will reference "male_rms English (United States)".
4. Use a text editing app to open the `voices-20120731.list` (Fig. 20).

   This file contains all information about the list of downloadable voices (e.g., Language, Country, Variant, MD5 checksum, etc.).

   The precise location of this file may slightly vary depending on the device.

   On this particular device it was located in the following:

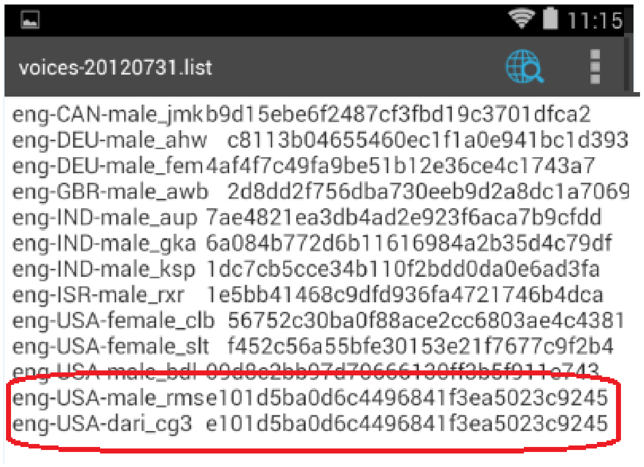
   ```
   /sdcard/flite-data/cg/voices-20120731.list
   ```



```
eng
1 item                      drw          3/27/2014
voices-20120731.list
654.00 B                    -rw          5/9/2014
```

Fig. 20 voices-20120731.list located in /sdcard/flite-data/cg

5. Copy the line that defines the "male_rms English (United States)" voice and paste it at the end of the file. On the line that was just pasted, replace the string "male_rms" with "dari_cg3". Save and close the file. Both "male_rms" and "dari_cg3" will share the same MD5 hash value (Fig. 21).



```
eng-CAN-male_jmk b9d15ebe6f2487cf3fbd19c3701dfca2
eng-DEU-male_ahw   c8113b04655460ec1f1a0e941bc1d393
eng-DEU-male_fem 4af4f7c49fa9be51b12e36ce4c1743a7
eng-GBR-male_awb   2d8dd2f756dba730eeb9d2a8dc1a7069
eng-IND-male_aup 7ae4821ea3db4ad2e923f6aca7b9cfdd
eng-IND-male_gka 6a084b772d6b11616984a2b35d4c79df
eng-IND-male_ksp 1dc7cb5cce34b110f2bdd0da0e6ad3fa
eng-ISR-male_rxr   1e5bb41468c9dfd936fa4721746b4dca
eng-USA-female_clb  56752c30ba0f88ace2cc6803ae4c4381
eng-USA-female_slt   f452c56a55bfe30153e21f7677c9f2b4
eng-USA-male_bdl 09d8e2bb07d70666130ff3b5f911e743
eng-USA-male_rms e101d5ba0d6c4496841f3ea5023c9245
eng-USA-dari_cg3  e101d5ba0d6c4496841f3ea5023c9245
```

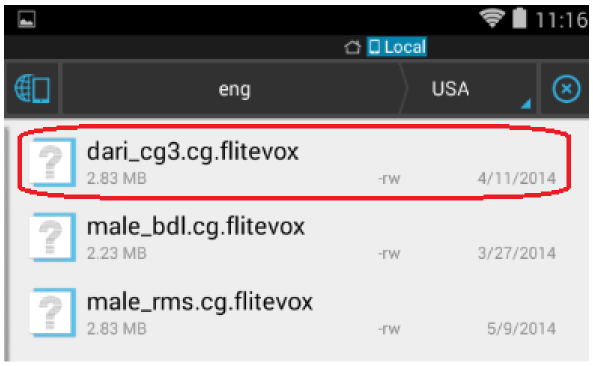
Fig. 21 Add an entry for "dari_cg3"

6. Use a folder browsing app to go to the downloaded file from step 3.

   On this particular device it was located on the following:

   ```
   /sdcard/flite-data/cg/eng/USA/male_rms.cg.flitevox
   ```

   Copy "male_rms.cg.flitevox" and save it as "dari_cg3.cg.flitevox" (Fig. 22).



dari_cg3.cg.flitevox 2.83 MB -rw 4/11/2014
male_bdl.cg.flitevox 2.23 MB -rw 3/27/2014
male_rms.cg.flitevox 2.83 MB -rw 5/9/2014

Fig. 22 Copy “male_rms.cg.flitevox” and save as “dari_cg3.cg.flitevox”

7. Intercept request for “dari_cg3” voice synthesis.

   This code update was made in the previous section. It is repeated here to simply explain how “dari_cg3.cg.flitevox” file is used to update the Flite Android Engine GUI.

   Add the following code at line 132 (actual line number may vary) in the method:

   `onSynthesizeText(SynthesisRequest,SynthesisCallback)` in FliteTtsService.java.

   It’s very important to use the same Language, Country, and Variant parameters used to add the Dari Clustergen to the voice list in the edu_cmu_cs_speech_tts_flite_engine.cc file. In earlier section, we defined Dari Clustergen by specifying “cmu_dari_lang” as Language, “AFG” as Country, and “dari_cg3” as Variant.

   Before edit:

```
if (! ((mLanguage == language) &&
       (mCountry == country) &&
       (mVariant == variant ))) {
       result = mEngine.setLanguage(language, country, variant);
. . .
}

if (!result) {
       Log.e(LOG_TAG, "Could not set language for synthesis");
```

   After edit:

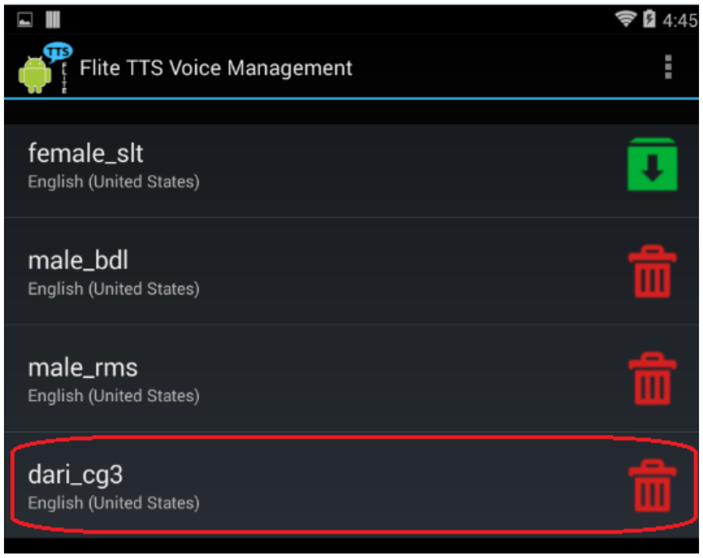
```
if (! ((mLanguage == language) &&
       (mCountry == country) &&
       (mVariant == variant ))) {
       result = mEngine.setLanguage(language, country, variant);
. . .
}

if(variant.equals("dari_cg3"))
       result = mEngine.setLanguage("cmu_dari_lang", "AFG", "dari_cg3");

if (!result) {
       Log.e(LOG_TAG, "Could not set language for synthesis");
```

8. Compile and run the app.

   The Flite TTS Voice Management page will now list “dari_cg3” as a selectable option (Fig. 23). Note that Management page shows “English (United States)” associated with “dari_cg3”. This is a side effect of using “male_rms” English voice as a source of the Dari surrogate Flitevox file. The English indicator can be safely ignored. Management page also shows that the voice is already downloaded, because the `dari_cg3.cg.flitevox` exists on the device.



4:45
Flite TTS Voice Management

female_slt
English (United States)

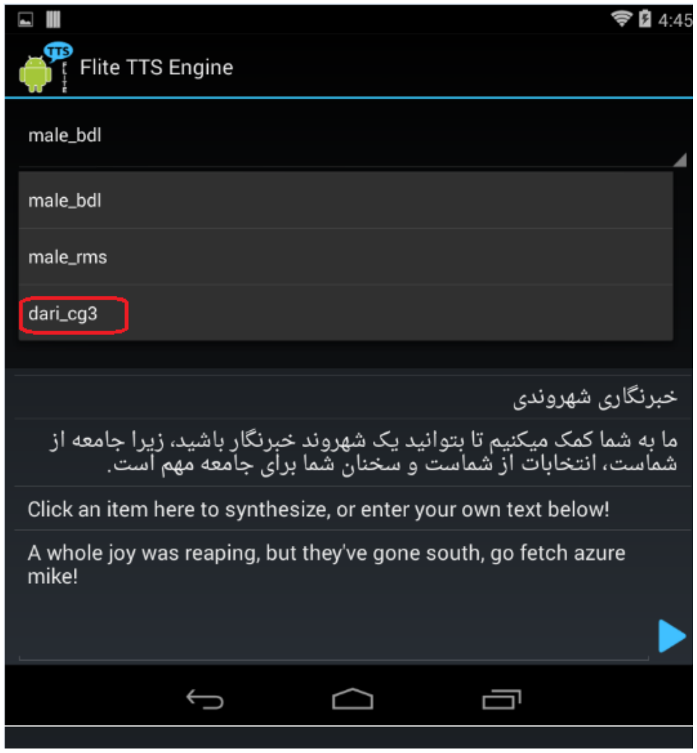
male_bdl
English (United States)

male_rms
English (United States)

dari_cg3
English (United States)

Fig. 23   Dari listed as selectable voice in the Flite TTS Voice Management page

The Flite TTS Engine page will list “dari_cg3” as a selectable option (Fig. 24).



male_bdl

male_bdl

male_rms

dari_cg3

خبرنگاری شهروندی

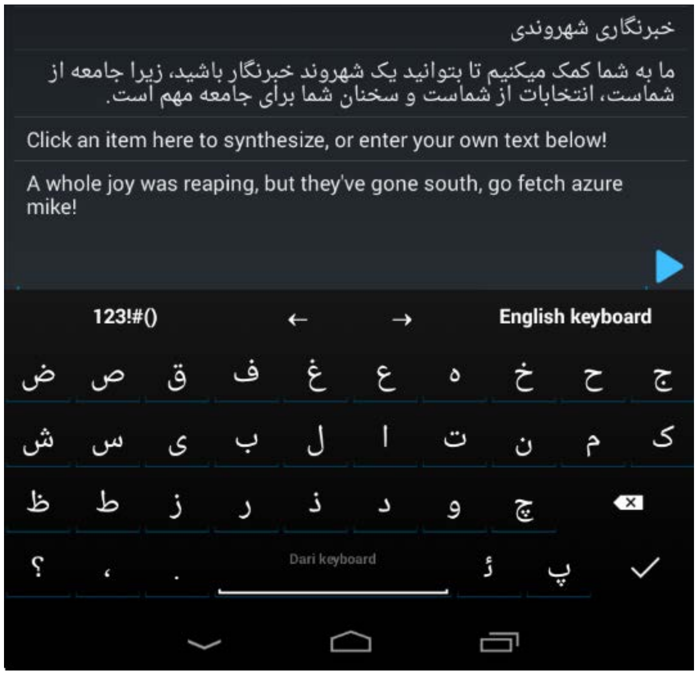
ما به شما کمک میکنیم تا بتوانید یک شهروند خبرنگار باشید، زیرا جامعه از شماست، انتخابات از شماست و سخنان شما برای جامعه مهم است.

Click an item here to synthesize, or enter your own text below!

A whole joy was reaping, but they've gone south, go fetch azure mike!

Fig. 24 Dari listed as selectable voice in the Flite Demo app

Now select “dari_cg3” as the Flite engine. Click on a pre-populated sample Dari text to hear the voice, or use a Dari keyboard (Fig. 25) to enter any Dari text.



خبرنگاری شهروندی

ما به شما کمک میکنیم تا بتوانید یک شهروند خبرنگار باشید، زیرا جامعه از شماست، انتخابات از شماست و سخنان شما برای جامعه مهم است.

Click an item here to synthesize, or enter your own text below!

A whole joy was reaping, but they've gone south, go fetch azure mike!

Fig. 25 Using a Dari keyboard to type Dari text

Dari Clustergen is now a selectable voice from the Flite Android Engine.

10. Invoking Flite Voices from an External Android App

Once the Android Flite Engine is able to invoke a Dari voice, it is ready to integrate with any Android app. Fortunately, the Android Flite Engine was designed following the Android TTS engine guideline. As a result, invoking a Flite voice from an external app is relatively straightforward. The Android Flite Engine's project page very briefly describes this process:

> *"If you are developing an application and would like to use Flite for speech synthesis, you can specify 'edu.cmu.cs.speech.tts.flite' as the package name of the engine to use."*[10]

This instruction implies that a developer could invoke the Flite Engine regardless of which TTS engine was set as default on the device. However, we encountered a problem with this approach and found that a device's default TTS engine setting played an important role for this task. For example, the Flite Dari voice method invocation did not work when the device's preferred TTS engine was set to "Google Text-to-Speech Engine", even though the *TextToSpeech* object accepted the "edu.cmu.cs.speech.tts.flite" package name without throwing an exception. In fact, the "setEngineByPackageName" method returned "SUCCESS" and there were promising status messages in the log, but it did not synthesize the Flite voice.

In order to invoke the Flite Dari voice, the device's default TTS engine had to be set to "Flite TTS Engine". Then, it was no longer necessary to set "edu.cmu.cs.speech.tts.flite" package name on the *TextToSpeech* object. As a side note, Google may be discouraging developers from directly specifying a TTS engine using a package name, because the "setEngineByPackageName" method has been deprecated since Android API 14. The downside to this approach is that "Flite TTS Engine" must be the default TTS engine (i.e., the developer's Android app will not fully function unless the device's preferred TTS engine is set to "Flite TTS Engine").

What follows is a step-by-step instruction on how to invoke Flite Dari voice from a separate Android app.

1. Configure device's preferred TTS engine (Fig. 26) to "Flite TTS Engine":

   Device Settings -> Language & input -> Test-to-speech output -> PREFERRED ENGINE -> choose "Flite TTS Engine"

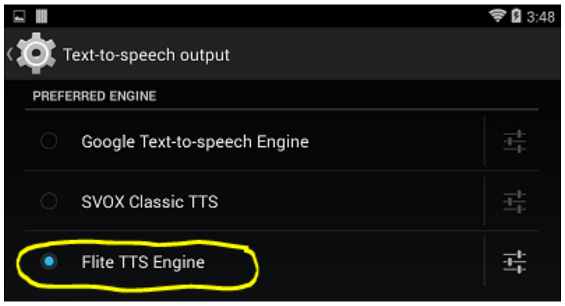


Fig. 26 Device's preferred TTS engine settings page

2. Instantiate a *TextToSpeech* object in the Activity's Java class. The mTextToSpeech object will assume it is using Flite TTS Engine and "edu.cmu.cs.speech.tts.flite" does not need to be set as the TTS Engine package name.

   The Dari Clustergen voice is specified by the Locale object. Use the same Language, Country, and Variant string used to name the Flitevox placeholder file (Fig. 27) to create a Locale object. Recall, "eng" was used as Language, "USA" was used as Country, and "dari_cg3" was used as Variant. Full path of the "dari_cg3.cg.flitevox" file on the Android device can be used to help identify the correct Language, Country, and Variant values to use.

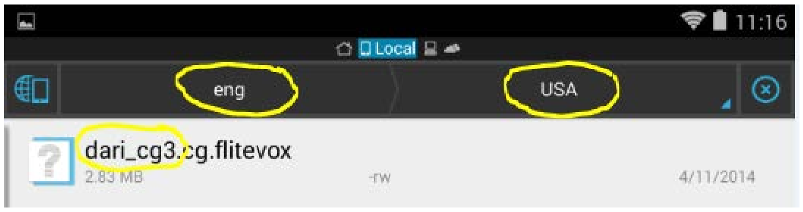


Fig. 27 Flitevox placeholder file for Dari Clustergen. Language, Country, and Variant values found in the file path

   The following is a sample of the line involved:

```
protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       private TextToSpeech mTextToSpeech = new TextToSpeech(this, new
OnInitListener() {
              public void onInit(int status) {
                 if(status == TextToSpeech.SUCCESS){
                     int result = mTextToSpeech.setLanguage(new
Locale("eng","USA", "dari_cg3"));
                     if( result == TextToSpeech.LANG_MISSING_DATA ||
                            result == TextToSpeech.LANG_NOT_SUPPORTED){
                        Log.d("TTS","This Language is not supported by the Text-
to-Speech engine.");
                     }
                 }else{
                     Log.d("TTS","Could not initialize Text-to-Speech engine.");
                 }
```

The *TextToSpeech* object, *mTextToSpeech*, exposes methods to control pitch and speech rate. These options would have been valuable to dynamically adjust the voice, but unfortunately they did not appear to affect the Flite Dari voice. We attempted to reduce the speech rate by 50%, but the synthesized Dari Voice remained unchanged.

3. Invoke the speak method. Once the *TextToSpeech* object is configured, synthesizing Dari text is easily accomplished by single line of code:

```
mTextToSpeech.speak("خبرنگاری شهروندی", TextToSpeech.QUEUE_FLUSH, null);
```

11. Conclusion

This report described all the steps required to integrate a Dari Clustergen voice to the Flite Android Engine. This lengthy and convoluted process involved transferring Dari from Flite 1.9.0 to Flite 1.5.6, making changes to the project source code, and then compiling Flite for multiple CPU targets.

If, in the future, Dari is available in Flitevox format, then there would be no need for these integration instructions. A user would simply download the Dari Flitevox file to their device and the Flite Android Engine would automatically list Dari as a selectable language and know how to synthesize Dari text. This is exactly how the Flite Android Engine currently works for the 12 English voices. Unfortunately, Dari Clustergen is not available in the Flitevox format.

Even though this report focuses on migrating Dari Clustergen to Android, it applies to other voices as well. In addition, this report described numerous problems we encountered during the migration process and workaround solutions that may help some developers doing similar work. In the end, our migration effort was successful and we were able to extend our work to synthesize the Dari Clustergen voice from our Dari Phrase Book Android app.

12. References

1. Winkler Robert, Metu Somiya, LaRocca Steve, Jahed Ghulam. English-to-Dari and Dari-to-English Medical Phrasebook Android Application Software System Documentation; Adelphi (MD): US Army Research Laboratory (US); 2012. Report No.: ARL-TR-5975.
2. Flite TTS Engine for Android description page. https://github.com/happyalu/Flite-TTS-Engine-for-Android/#introduction (accessed September 2014).
3. Hunt A, Black A. Unit selection in a concatenative speech synthesis system using a large speech database. in Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings, 1996 IEEE International Conference on, vol. 1, May 1996, pp. 373–376 vol. 1.
4. Black A, Zen H, Tokuda K. Statistical parametric speech synthesis. Speech Communication. 2009;51(11):1039–1064.
5. Email address for Dr Alan Black: awb@cs.cmu.edu.
6. Flite 1.5.6. download page. http://tts.speech.cs.cmu.edu/aup/distr/android/flite-1.5.6-current.tar.bz2 (accessed September 2014).
7. Android Development Tool download page. http://developer.android.com/sdk/index.html (accessed September 2014).
8. Android Native Development Kit download page. http://developer.android.com/tools/sdk/ndk/index.html (accessed September 2014).
9. Flite TTS Engine for Android project home. https://github.com/happyalu/Flite-TTS-Engine-for-Android#flite-tts-engine-for-android/ (accessed September 2014).
10. Using Flite for TTS in Your Application. https://github.com/happyalu/Flite-TTS-Engine-for-Android#using-flite-for-tts-in-your-application (accessed September 2014).

List of Symbols, Abbreviations, and Acronyms

ADT	Android Development Tool
ANT	Another Neat Tool
API	application programming interface
ARL	US Army Research Laboratory
CMU	Carnegie Mellon University
CPU	central processing unit
Flite	Festival Lite
GUI	graphical user interface
IDE	Integrated Development Environment
NDK	Native Development Kit
TTS	text-to-speech

1 (PDF)	DEFENSE TECH INFO CTR ATTN DTIC OCA
2 (PDF)	US ARMY RSRCH LABORATORY ATTN IMAL HRA MAIL & RECORDS MGMT ATTN RDRL CIO LL TECHL LIB
1 (PDF)	GOVT PRNTG OFC ATTN A MALHOTRA
3 (PDF)	US ARMY RSRCH LAB ATTN RDRL CII B MICHAEL LEE ATTN RDRL CII B R WINKLER ATTN RDRL CII B S LAROCCA

INTENTIONALLY LEFT BLANK.